VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

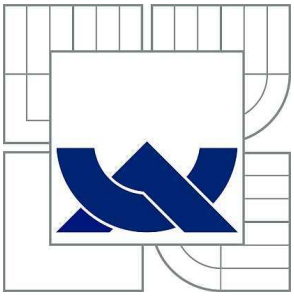
SOFTWARE PRO SIMULACI ŘÍDICÍCH ALGORITMŮ S GENEROVÁNÍM C-KÓDU PRO MIKROKONTROLÉR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

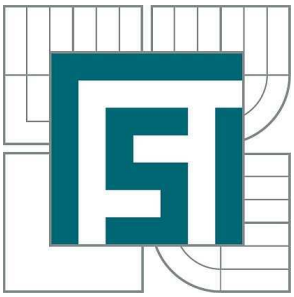
MARTIN FIALA

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ

ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

FACULTY OF MECHANICAL ENGINEERING

INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

SOFTWARE PRO SIMULACI ŘÍDICÍCH ALGORITMŮ S GENEROVÁNÍM C-KÓDU PRO MIKROKONTROLÉR

SOFTWARE FOR SIMULATION OF CONTROL ALGORITHMS WITH C CODE GENERATION FOR MICROCONTROLLER

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — MARTIN FIALA

AUTHOR

VEDOUCÍ PRÁCE — Ing. ROBERT GREPL, Ph.D.

SUPERVISOR

BRNO 2010

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav mechaniky těles, mechatroniky a biomechaniky
Akademický rok: 2009/2010

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Martin Fiala

který/která studuje v **bakalářském studijním programu**

obor: **Mechatronika (3906R001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Software pro simulaci řídicích algoritmů s generováním C-kódu pro mikrokontrolér

v anglickém jazyce:

Software for simulation of control algorithms with C code generation for microcontroller

Stručná charakteristika problematiky úkolu:

Práce se bude zabývat návrhem a implementací vlastního prostředí pro simulaci, testování a generování kódu pro mechatronické aplikace. Předpokládáme využití open-source prostředí (např. jazyk Python).

Cíle bakalářské práce:

1) Rešerše v tématických oblastech: mikrokontroléry dsPIC, automatické generování C kódu pro vestavěné řídicí aplikace z jazyků vyšší úrovně.
2) Návrh struktury programu pro model řízené soustavy v jazyce Python (explicitní ODE, integrace metodou Runge-Kutta). Možnost propojení na externí fyzikální engine (např. ODE nebo Havok). Vypracování několika konkrétních příkladů (SISO i MIMO). Verifikace výsledků pomocí Matlab/Simulink.
3) Návrh struktury softwaru s uvažováním všech prvků regulátoru (sensorika, zpětnovazební regulátor, feedforward blok, porucha, šum sensorů, stavový odhad, generátor referenční trajektorie). Implementace v jazyce Python. Možnost specifikace datových typů jednotlivých signálů a různého časování bloků. Ukázka simulace celé řízené soustavy na konkrétních příkladech, verifikace pomocí modelu v Matlabu.
4) Generování C kódu pro SIL. Jsou respektovány datové typy reálného mikrokontroléru dsPIC nebo PIC32. Vygenerovaný a zkompilovaný kód je simulován na PC společně s modelem soustavy.
5) Řízení reálné soustavy. Vygenerovaný kód je nahrán přímo do mikrokontroléru, který řídí reálnou soustavu. Pomocí sériového rozhraní je možné získat informace o vybraných signálech regulátoru.
6) Produkční kód. Jsou odstraněny veškeré pomocné části kódu. V práci bude podrobně

zdokumentováno několik příkladů použití vytvořeného softwaru na konkrétních příkladech řízení reálných soustav (DC motor, škrticí klapka, model vrtulníku).

Seznam odborné literatury:

- Herout, P.: Učebnice jazyka C
- Harms: Začínáme programovat v jazyce Python, 2003
Kiusalaas, J.: Numerical method in engineering with Python, 2005
- Mann, B.: C pro mikrokontroléry, Nakladatelství BEN, 2003
- microchip.com

Vedoucí bakalářské práce: Ing. Robert Grepl, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2009/2010.

V Brně, dne 5.11.2009

L.S.

prof. Ing. Jindřich Petruška, CSc. Ředitel ústavu	prof. RNDr. Miroslav Doupovec, CSc. Děkan fakulty

Abstrakt

Tato práce se zabývá simulací průběhu signálu v zadané jednoduché struktuře s jedním regulátorem, řízenou soustavou a zpětnou vazbou pomocí open-source programovacího jazyka Python. V další části je rozebíráno generování C kódu regulátoru pro mikrokontroléry a následnou kontrolu průběhu signálu, který mikrokontrolér zpracovává. Tato kontrola je prováděna pomocí sériového rozhraní.

Abstract

This work is concerned with simulation of signal processing in a simple scheme with one controller, plant and feedback by open-source programming language Python. Moreover it is concerned with generating C-code for microcontrollers and in consequent check up of signal processing which is processed by microcontroller. This check up is done by serial interface.

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Robertu Greplovi Ph.D. za cenné rady, připomínky a metodické vedení práce.

Prohlášení:

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně na základě svých vědomostí, studia literárních zdrojů a pokynů vedoucího mé práce.

V Brně dne:

……
Martin Fiala

OBSAH:

1 ÚVOD

Většina profesionálních aplikací pro simulaci a řízení mechatronických soustav je vytvářena v prostředí MATLAB/Simulink[20] vyvíjeném společností MathWorks. Výhody tohoto použití jsou možnost vytváření a řízení složitých soustav a také softwarová podpora ze strany velkého množství firem. Naopak velká nevýhoda spočívá v celkové vysoké ceně tohoto řešení. Program MATLAB by byl zastupitelný nějakým volně šiřitelným nástrojem pro matematické výpočty, například programem Octave[21], ale nástroj Simulink pro simulaci průběhů signálů nelze tak snadno nahradit.

Záměrem této práce je vytvořit volně šiřitelný program pro simulaci průběhu signálu na jednoduchém modelu, který je tvořen regulátorem, řízenou soustavu, zpětnou vazbou, šumem senzorů a generátorem referenční trajektorie.

Dalším cílem je usnadnění vytváření řídících aplikací pro mikrokontroléry tím, že se bude generovat C-kód regulátoru, kterým bude možné mikrokontroléry naprogramovat a následně odsimulovat jejich řízení na sestaveném modelu.

K tomuto účelu je vhodné využít programovací jazyk Python.

2 PYTHON

2.1 Python

Python[3] je volně šiřitelný programovací jazyk, který je nezávislý na platformě. Běží tedy na systémech Windows, Linux, Mac OS a dalších. Vývoj programů v tomto jazyce probíhá rychleji než v tradičních jazycích, například v C nebo C++. Jeho zápis je oproti nim značně zjednodušen a zpřehledněn. Lze v něm vytvářet jak spustitelné aplikace, tak i skripty, které vyžadují instalaci kompletního programu Python.[1]

Aby se v práci tento jazyk lépe využil, byla pro matematické výpočty použita řada volně šiřitelných knihoven, které rozšiřují jeho funkce.

Knihovny, které byly použity při programování této práce, jsou popsané v následujících kapitolách.

2.2 NumPy

NumPy[5] je matematická knihovna, která je potřebná k vědeckým numerickým výpočtům v Pythonu. Obsahuje funkce pro práci s maticemi, převody mezi různými typy dat a polí a mnoho dalších matematických funkcí, například Fourierovu transformaci nebo generování náhodných čísel.

2.2.1 Použité funkce

numpy.ndarray

Pro zjednodušení maticových výpočtů je v programu použit datový objekt n-rozměrné pole (ndarray) z knihovny NumPy, se kterým lze provádět matematické operace stejné jako s maticí (maticové násobení, dělení, sčítání a odečítání matic, atd.).[7],[8]

numpy.rot90(A)

Pro rotaci matic slouží funkce rot90, která je přetáčí o 90°. Tato funkce je použita pro převod řádkových vektorů na sloupcové, aby je bylo možné navzájem vynásobit. Vstupem i výstupem je typ ndarray.

numpy.dot(A, B)

Tato funkce slouží pro vynásobení dvou matic. Vstupy *A* a *B* jsou typu ndarray, výstupem je také typ ndarray.

2.3 SciPy

SciPy[9] znamená „Scientific Tools for Python“, neboli vědecké nástroje pro Python. Je to volně šiřitelná a jednoduše použitelná knihovna pro tento jazyk, která využívá pro matematické a vědecké výpočty, například pro numerickou integraci, optimalizaci nebo pro práci s přenosy signálů. Je ovšem propojená s knihovnou NumPy (viz kapitola 2.2), ze které používá pro výpočty n-rozměrné pole (typ ndarray) a operace pro práci s ním.

2.3.1 Použité funkce

scipy.signal.tf2ss(num, den)

Funkce slouží pro převod přenosové funkce na matice stavové rovnice. Vstup *num* znázorňuje čitatel a vstup *den* jmenovatel polynomu přenosové funkce. Výstupem jsou matice vyjadřující dynamiku stavového systému – *A, B, C, D*.[11]

scipy.integrate.odeint(func, y0, t, args = ())

Funkce slouží k integraci soustavy diferenciálních rovnic. Vstup *func* znázorňuje soustavu diferenciálních rovnic, *y0* počáteční hodnotu, *t* pole času, ve kterém se soustava integruje a *args* představují odkazy na další potřebné proměnné, které se vyskytují v soustavě diferenciálních rovnic.

2.4 MatPlotLib

MatPlotLib[13] je knihovna pro vykreslování 2D grafů, které lze následně uložit do souboru jako obrázek. Lze s ním vykreslit funkce, histogramy, barevná spektra nebo i sloupcové diagramy.

V projektu je použita funkce *pyplot* k znázornění průběhu signálu skrz soustavu.

2.4.1 Použité funkce

matplotlib.pyplot.plot(x, y)

Funkce vykresluje graf, kde *x* znázorňuje pole hodnot osy x a *y* pole hodnot osy y.[14]

matplotlib.pyplot.savefig(FileName)

Funkce ukládá vykreslený graf do souboru, ke kterému je cesta *FileName*.

2.5 pySerial

PySerial[16] je knihovna umožňující jednodušší přístup a komunikaci po sériovém portu počítače.

3 DSPIC A GENEROVÁNÍ C-KÓDU

Mikrokontroléry dsPIC[18] jsou 16-bitové programovatelné polovodičové součástky vyráběné firmou Microchip, které se používají k řízení různých procesů. Pracují na architektuře RISC. Mají tedy menší počet instrukcí a pevně danou šířku slova instrukce. Jejich programová a datová paměť je oddělená. Často se používají pro řízení různých jednoduchých aplikací, například motorů.

Aby byly mikrokontroléry použitelné pro danou aplikaci řízení, musejí se nejprve naprogramovat. K tomuto účelu se nejvíce využívá jazyk C. Je to univerzální jazyk nízké úrovně. To znamená, že přímo pracuje pouze se základními datovými typy, jako jsou znaky a celá a reálná čísla, není specializovaný na jednu určitou oblast a má velice blízko k assembleru. Proto například i program Matlab, který je určen převážně k vědeckým výpočtům a pro práci s maticemi, umožňuje generovat své skripty do jazyka C. A program z této práce také generuje pro usnadnění programování mikrokontrolérů kód pro tento jazyk.

Mikrokontroléry dsPIC ovšem oproti výpočtům v simulaci hardwarově nepodporují práci s plovoucí desetinnou čárkou. Kdyby se naprogramovaly kódem pracujícím v této aritmetice, trval by výpočet velmi dlouhou dobu. Mikrokontrolér dsPIC podporuje pouze celočíselnou aritmetiku, která bohužel na praktické aplikace většinou nestačí. Proto se u takovýchto zařízení používá aritmetika založená na pevné desetinné čárce, která umožňuje uchovat i desetinné číslo. Stačí, aby bylo v programu určeno, od kterého bitu číslo představuje hodnotu za desetinnou čárkou.

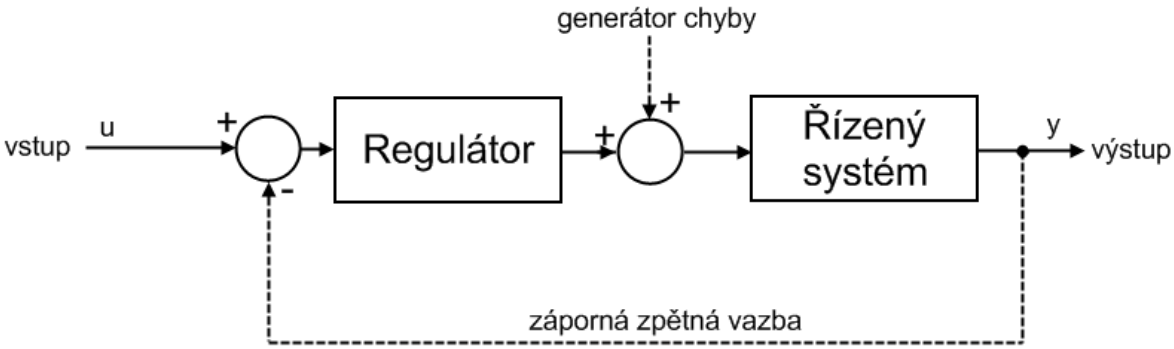
Řízení těmito mikrokontroléry pak většinou probíhá tak, že se periodicky po pevně daném čase načte vstupní hodnota, kterou může být digitální či analogový signál ze senzoru, který se pak následně převádí na požadovanou číselnou hodnotu. Ta se dále zpracovává daným výpočtem regulátoru a odešle se na výstup, nejčastěji ve formě PWM signálu.

4 VLASTNÍ PROGRAM

4.1 Použití

Pomocí programu z této práce lze simulovat průběh signálu skrz soustavu, která je znázorněna na Obr. 1. Přestože je tento systém jednoduchý, v mnoha použitích zcela postačuje. Soustava obsahuje regulátor, řízený systém, zpětnou vazbu, kterou lze zapnout či vypnout, a generátor chyby. Pro mikrokontrolér lze následně vygenerovat podprogram regulátoru ve formě C-kódu, pokud je přenos skrz něj ve tvaru diskrétní přenosové funkce či PID regulátoru.

Regulátor či řízený systém lze nahradit komunikací po sériovém portu s libovolným zařízením, které přijímá vypočtené hodnoty typu float nebo integer a vrací nazpět přepočítané hodnoty výstupu. Toto lze použít například při práci s reálným regulátorem, když chceme zkontrolovat jeho správnou funkci.



Obr. 1 – schéma použití programu

Simulátor je naprogramován ve formě skriptu pro Python, tudíž pro jeho použití je nutné mít Python nainstalovaný.

4.2 Instalace

Pro správnou funkci programu je nutné mít nainstalovaný Python verze 2.6[4] a knihovny NumPy[6], SciPy[10], MatPlotLib[15] a pySerial[17], které se dají volně stáhnout na internetu.

Funkce pro výpočet a vykreslení průběhu signálu a pro generování C-kódu pro mikrokontroléry dsPIC se nachází v souboru *Simulace.py*. Tento soubor je třeba nakopírovat do složky programu Python do podsložky „*.\Lib\site-packages*“ (např. „*C:\Program Files\Python265\Lib\site-packages*“).

Při vytváření vlastního skriptu pro Python stačí pro použití těchto funkcí tento soubor importovat:

```
from Simulace import *
```

4.3 Popis funkcí

time(tmax, step)

Funkce generuje řadu čísel typu float, které začínají hodnotou 0, postupně se navyšují o hodnotu *step* až do hodnoty *tmax*. Používá se na vygenerování tabulky časů, podle které se integruje průběh signálu.

Příklad použití:

```
t = time(10, 0.1)

>> t = [0.0, 0.1, 0.2, ... , 9.8, 9.9, 10.0]
```

step(t, start, max = 1.0, min = 0.0, typ = ‘float‘)

Funkce vygeneruje průběh jednotkového skoku pro tabulku času *t*, kdy jednotkový skok se provede v čase *start*. Nepovinný parametr *max* vyjadřuje velikost konečné hodnoty, která je defaultně nastavená na velikost 1. Nepovinný parametr *min* vyjadřuje velikost původní hodnoty a je defaultně nastavená na 0. Nepovinný parametr *typ* určuje, jakého typu budou generované prvky, zda-li čísla s plovoucí desetinnou čárkou ‘*float*‘, či celá čísla ‘*int*‘. Je také možné generovat signál s více vstupy tak, že za parametry *start*, *max* a *min* nebudeme dosazovat samostatné prvky, ale pole s příslušným počtem prvků, kolik je potřeba výstupů.

Příklad použití – generování jednoho vstupu:

```
t = time(10, 0.1)
u = step(t, 1)

>> t = [0.0, 0.1, ... , 0.9, 1.0, 1.1, ..., 9.9, 10.0]
>> u = [0.0, 0.0, ... , 0.0, 1.0, 1.0, ..., 1.0, 1.0]
```

Příklad použití – generování více vstupů:

```
t = time(10, 1)
u = step(t, [1, 2], [2.0, 1.0])

>> t = [ 0.0,         1.0,         2.0,        ..., 10.0]
>> u = [[0.0, 0.0], [2.0, 0.0], [2.0, 1.0], ..., [2.0, 1.0]]
```

sine(t, period, amplitude = 1.0, shift = 0.0, typ = ‘float‘)

Funkce vygeneruje průběh sinusového signálu pro tabulku času *t*, parametr *period* udává velikost periody sinusového průběhu. Nepovinný parametr *amplitude* udává velikost amplitudy signálu, která je defaultně nastavena na 1. Nepovinný parametr *shift* udává posunutí v časové ose. Nepovinný parametr *typ* určuje, jakého typu budou generovány prvky, zda-li čísla s plovoucí desetinnou čárkou ‘*float*‘, či celá čísla ‘*int*‘. Je také možné generovat signál s více vstupy tak, že za parametry *period*, *amplitude* a *shift* nebudeme dosazovat samostatné prvky, ale pole s příslušným počtem prvků, kolik je potřeba výstupů.

Příklad použití:

```
t = time(10, 0.1)
u = step(t, 1)
```

```
>> t = [0.0, 0.1,              ..., 9.9,              10.0]
>> u = [0.0, 0.5877852522924, ..., -0.5877852522924, 0.0]
```

pulse(t, period, width, saw = False, amplitude = 1.0, shift = 0.0, typ = ‘float‘)

Funkce vygeneruje průběh PWM signálu pro tabulku času *t*. Proměnná *period* udává velikost periody pulzního průběhu, proměnná *width* délku sepnutého stavu signálu. Funkce má nepovinný parametr *saw*. Pokud nabývá hodnoty True, pak funkce generuje pilový průběh, pokud nabývá hodnoty False, generuje průběh obdélníkový. Nepovinný parametr *amplitude* znázorňuje výšku hrany signálu a nepovinný parametr *shift* znázorňuje posun v časové ose. Nepovinný parametr *typ* určuje, jakého typu budou generovány prvky, zda-li čísla s plovoucí desetinnou čárkou ‘*float*‘, či celá čísla ‘*int*‘. Je také možné generovat signál s více vstupy tak, že za parametry *period*, *width*, *saw*, *amplitude* a *shift* nebudeme dosazovat samostatné prvky, ale pole s příslušným počtem prvků, kolik je potřeba výstupů.

Příklad použití – generování obdélníku s jedním výstupem:

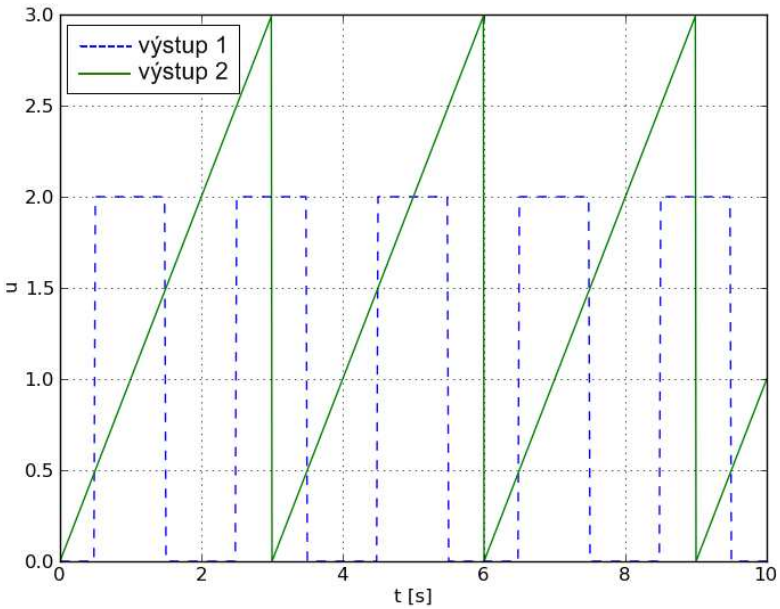
```
t = time(10, 0.1)
u = pulse(t, 0.4, 0.2)

>> t = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, ...]
>> u = [1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, ...]
```

Příklad použití – generování pily:

```
t = time(10, 0.1)
u = pulse(t, 0.4, 0.4,True)

>> t = [0.0, 0.1,  0.2, 0.3,  0.4, 0.5,  ...]
>> u = [0.0, 0.25, 0.5, 0.75, 0.0, 0.25, ...]
```

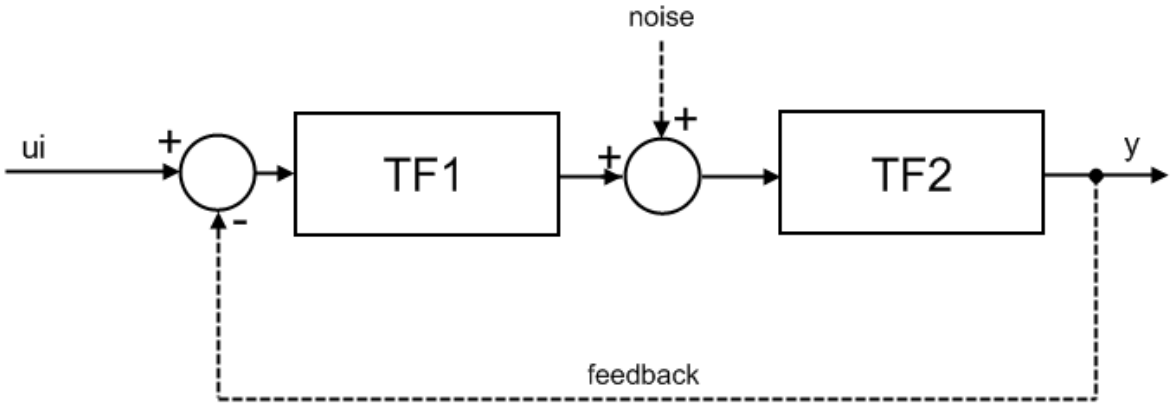
Příklad použití – generování signálu s více výstupy (viz. Obr. 2):

```
t = time(10, 0.01)
u = pulse(t,[2,3],[1,3],[False,True],[2.0,3.0],[0.5,0.0])
plot(t,u)
```



Obr. 2 – Znázornění funkce pulse s více výstupy

transfer(TF1, TF2, t, ui, feedback = False, x0 = [[0.0], [0.0]], noise = False, noiseVal = [-0.01,0.01] , typ = [‘float‘, ‘float‘])



Obr. 3 – Znázornění funkce transfer

Funkce vypočítává průběh signálu přes soustavu znázorněnou na Obr. 3. Parametry *TF1* a *TF2* znázorňují přenosy a mohou nabývat těchto formátů:

1. Přenosová funkce – zadává se jako pole se dvěma prvky znázorňující čitatele a jmenovatele zlomku přenosové funkce.

   Příklad použití – zápis přenosové funkce:

   $$F(p) = \frac{1}{5p^2 + 8p + 3} \qquad (1)$$

   – v Pythonu:

   ```
   TF = [[1],[5, 8, 3]]
   ```

2. Matice stavové rovnice – zadává se jako pole se čtyřmi prvky, z nichž každý prvek znázorňuje jednu z matic stavové rovnice A,B,C,D.

   Příklad použití – zápis stavových matic:

   $$A = \begin{bmatrix} 0 & 1 \\ -0{,}6 & -1{,}6 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad C = \begin{bmatrix} 0{,}2 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 0 \end{bmatrix} \qquad (2)$$

   – v Pythonu:

   ```
   TF = [[[0,1],[-0.6,-1.6]], [[0],[1]], [0.2,0], [0]]
   ```

3. Diskrétní přenosová funkce – zadává se jako pole se třemi prvky, z nichž první dva znázorňují čitatele a jmenovatele zlomku přenosové funkce, a třetí slouží jako rozlišovací znamení od obyčejné přenosové funkce a může nabývat libovolné hodnoty.

   Příklad použití – zápis diskrétní přenosové funkce:

   $$F(z) = \frac{1}{5z^2 + 8z + 3} \qquad (3)$$

   – v Pythonu:

   ```
   TF = [[1],[5, 8, 3],‘z‘]
   ```

4. Nelineární soustava – zadává se jako odkaz na funkci, ve které je definována dynamika přenosu. Funkce musí být ve formátu *f(x, t, u)*, kde *f* znázorňuje název funkce, *x* je pole veličin, pro které se přenos počítá, *t* je čas a *u* je pole vstupů. Funkce musí navracet pole derivací veličin *dx*.

   Příklad použití – zápis nelineární soustavy

$$\begin{aligned} dx_1 &= x_2 \\ dx_2 &= 6{,}8 \cdot x_1 + 8.0 \cdot x_2 + u \cdot \sin(2 \cdot t) \end{aligned} \tag{4}$$

   – v Pythonu:

```
def funkce(x,t,u):      # nejprve je nutné definovat funkci
   dx = [0.0]*2         # vytvoření výstupního pole
   dx[0] = x[1]
   dx[1] = x[0]*6.8 + x[1]*8.0 + u[0]*sin(2*t)
   return dx

TF = funkce       # pak lze na ni odkázat pro výpočet přenosu
```

5. PID regulátor – zadává se jako pole se třemi prvky – konstanta proporcionálního členu K_p, konstanta integračního členu K_i a konstanta derivačního členu K_d.

   Příklad použití – zápis PID regulátoru

$$K_p = 10, K_i = 50, K_d = 0{,}6. \tag{5}$$

   – v Pythonu:

```
TF = [10.0, 50.0, 0.6]
```

6. Komunikace po sériovém rozhraní – zadává se jako pole s jedním prvkem, který udává název sériového portu, po kterém program komunikuje. Odesílaná a přijímaná data jsou ve formátu textového řetězce, který vyjadřuje hodnotu čísla, zakončeného znakem pro konec řádku. Ten má podle ASCII tabulky číselnou hodnotu 10. Komunikace probíhá rychlostí 115200kbps.

   Příklad použití – komunikace na sériovém portu COM1:

```
TF = ['COM1']
```

Parametr *t* je pole časů, pro které se vypočítává průběh signálu, parametr *ui* je pole vstupů pro časy *t*, nepovinný parametr *feedback* je typu boolean a zapíná či vypíná zpětnou vazbu, defaultně je zpěná vazba vypnutá. Nepovinný parametr *x0* zadává počáteční podmínky pro přenosy regulátoru i systému. Nepovinný parametr *noise* zapíná šum za regulátorem, což je přičítání náhodných čísel k signálu v určitém rozsahu, které udává nepovinný parametr noiseVal. Ten je defaultně nastaven na generování náhodných čísel v rozsahu od -0,01 do 0,01. Nepovinný parametr *typ* určuje, s jakými čísly budou pracovat jednotlivé bločky systému, buď s čísly s plovoucí desetinnou čárkou '*float*', či s celými čísly '*int*'.

Pokud se výpočet zdaří, je výstupem funkce pole výstupních hodnot pro časy *t*. Pokud se ve výpočtu objeví nějaká chyba, je výstupem řetězec oznamující informace o vzniklé chybě.

plot(x, y, FileName = '', Title = 'Transfer', xLabel = 'time (s)', yLabel = 'transfer (-)')

Funkce vykresluje průběh signálu a ukládá ho do souboru jako obrázek. Parametr *x* znázorňuje hodnoty osy x, parametr *y* hodnoty osy y. Nepovinný parametr *FileName* znázorňuje název souboru, do kterého se má graf uložit. Pokud je řetězec prázdný, soubor se neukládá. Nepovinný parametr *Title* obsahuje název grafu, parametr *xLabel* obsahuje popisek osy x a parametr *yLabel* obsahuje popisek osy y.

Funkce umožňuje vykreslit i průběh 2 signálů, pokud budou obsaženy v parametru *y*. Mohou být vyjádřeny buď formou 2 spojených tabulek, nebo jako tabulka vektorů.

Příklad použití – možné vykreslení různých signálů:

```
t = [1,2,3,4]
y = [1,2,3,4]                    # 1) vykreslení jednoho průběhu
y = [[1,2,3,4], [5,3,1,8]]       # 2) vykreslení 2 průběhů
y = [[1,5],[2,3],[3,1],[4,8]]    # 3) 2 průběhy – jiná forma zápisu
plot(t,y)                        # vykreslení grafu
```

CreateC(File, TF, StepTime, Type = 'double')

Funkce generuje C-kód ve formě podprogramu pro výpočet výstupních hodnot, které jsou potřeba k řízení daného systému. Do parametru *TF* se zadáví buď přenos diskrétní přenosové funkce, nebo konstanty všech tří složek PID regulátoru. Délka kroku výpočtu se zadává v sekundách do parametru *StepTime*. C-kód se ukládá do souboru s cestou *File*.

Příklad použití – kompilace kódu diskrétní přenosové funkce a PID regulátoru:

```
TfZ = [[2.512, -2.5],[1, -1]]              # zadání diskrétní p.f.
CreateC('C:\main',TfZ,0.01,'double')       # kompilace – typ double

PID = [20,3,10]]                           # zadání PID regulátoru
CreateC ('C:\main',PID,0.01,'int')         # kompilace – typ integer
```

status(serial, cycles, input = True, output = True, signal = 0)

Funkce slouží ke zjištění aktuální hodnoty vstupu a výstupu na mikrokontroléru. Tyto hodnoty zjišťuje opakovaně, počet cyklů udává parametr *cycle*s. Pokud bude parametr *input* rovný hodnotě True, budou se zjišťovat vstupy, pokud bude parametr *output* rovný hodnotě True, budou se zjišťovat i výstupy. Nepovinný parametr *signal* vyjadřuje tabulku požadovaných vstupních hodnot veličiny, kterou mikrokontrolér řídí. Parametr *serial* udává název sériového rozhraní, po kterém program komunikuje s mikrokontrolérem.

Program vyšle pro zjištění vstupu znak 'u' a znak pro ukončení řádku, který má v ASCII tabulce hodnotu 10. Pak očekává odezvu od mikrokontroléru ve formě řetězce obsahujícího hodnotu vstupu a ukončeného zase znakem pro ukončení řádku. Při zjišťování výstupu komunikace probíhá obdobně, jen program vyšle místo znaku 'u' znak 'y'. Při změně požadované vstupní hodnoty program vyšle znak 'h' a za ním požadovanou hodnotu převedenou na řetězec se znakem pro ukončení řádku. Výstupem funkce je tabulka načtených hodnot, které lze dále vykreslit funkcí *plot*.

Příklad použití:

```
# zjistit 1000 hodnot jen z výstupu
y = status('COM1', 1000, False, True)
x = range(1000)                            # vytvoření pole osy x
plot(x,y)                                  # vykreslení hodnot
```

4.4 Příklady simulace

4.4.1 Průběh jednotkového skoku přes diskrétní regulátor

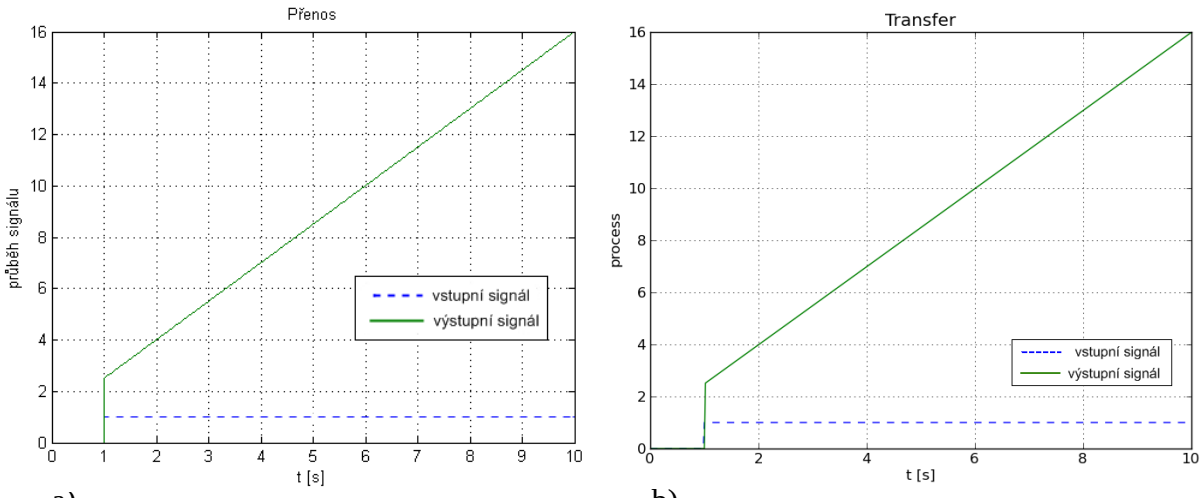
Pokud je potřeba simulovat jednotkový skok přes diskrétní regulátor s přenosovou funkcí

$$F(z) = \frac{2{,}515 - 2{,}5z^{-1}}{1 - 1z^{-1}}, \qquad (6)$$

je nutné nejprve vytvořit pole času, například s výpočetním krokem 0,01s do času 10s. Dále se vytvoří pole vstupů pro tyto časy s jednotkovým skokem v čase 1s. Teď je na řadě zapsat přenosovou funkci regulátoru do formátu pro Python a také zapsat jednotkový přenos pro řízený systém. Nakonec už zbývá jen pomocí funkce *transfer* vypočítat průběh signálu a příkazem *plot* signál vykreslit. Na Obr. 4 je vidět porovnání výpočtu v Simulinku a tomto programu.

Kód v Pythonu:

```
h = 0.01                        # nastavení výpočetního kroku 0.01s
tmax = 10                       # max. čas, do kterého trvá výpočet
t = time(tmax,h)                # vytvoření pole času
u = step(t,1)                   # vytvoření pole vstupů s jednotkovým
                                #  skokem v čase 1s

Tf1 = [[2.515, -2.5], [1, -1],'z']      # přenos přes regulátor
Tf2 = [[1], [1]]                        # přenos přes systém

y = transfer(Tf1,Tf2,t,u)               # výpočet výstupu
plot(t,y)                               # vykreslení grafu
```



a) b)

Obr. 4 – Průběh jednotkového skoku přes regulátor. a) V Simulinku. b) V tomto programu.

4.4.2 Průběh jednotkového skoku přes řízený systém

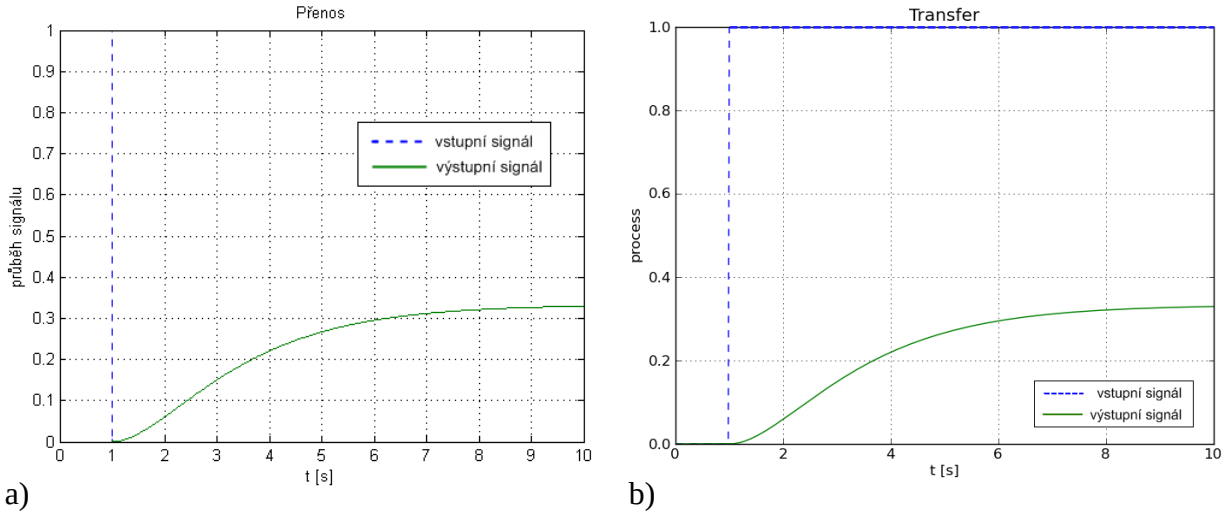
Pokud je potřeba simulovat jednotkový skok přes řízený systém, který má přenosovou funkci

$$F(p) = \frac{1}{5p^2 + 8p + 3}, \qquad (7)$$

postupuje se stejným způsobem jako v předcházejícím případě. Pouze za přenos regulátoru se dosadí jednotkový přenos a za přenos řízeného systému se dosadí převedená přenosová funkce ve formátu pro Python. Na Obr. 5 je vidět porovnání výpočtu v Simulinku a tomto programu.

Kód v Pythonu:

```
Tf1 = [[1],[1]]                 # přenos přes regulátor
Tf2 = [[1],[5, 8, 3]]           # přenos přes systém

y = transfer(Tf1,Tf2,t,u)       # výpočet výstupu
plot(t,y)                       # vykreslení grafu
```



a) b)

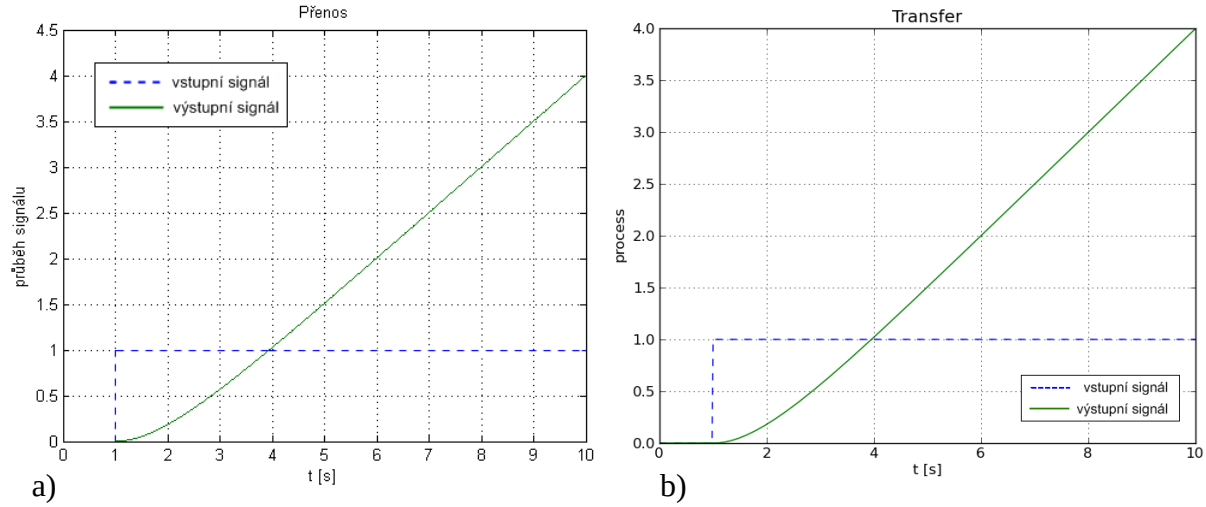
Obr. 5 – Průběh jednotkového skoku přes řízený systém. a) V Simulinku. b) V tomto programu.

4.4.3 Průběh jednotkového skoku soustavou bez zpětné vazby

Pokud je potřeba simulovat přenos celého tohoto systému bez zpětné vazby, dosadí se za přenos regulátoru a řízeného systému přenosové funkce z předchozích příkladů 4.4.1 a 4.4.2 ve formátu pro Python. Na Obr. 6 je vidět porovnání výpočtu v Simulinku a tomto programu.

Kód v Pythonu:

```
Tf1 = [[2.515, -2.5], [1, -1],'z']    # přenos přes regulátor
Tf2 = [[1],[5, 8, 3]]                 # přenos přes systém

y = transfer(Tf1,Tf2,t,u)             # výpočet výstupu
plot(t,y)                             # vykreslení grafu
```



a) b)

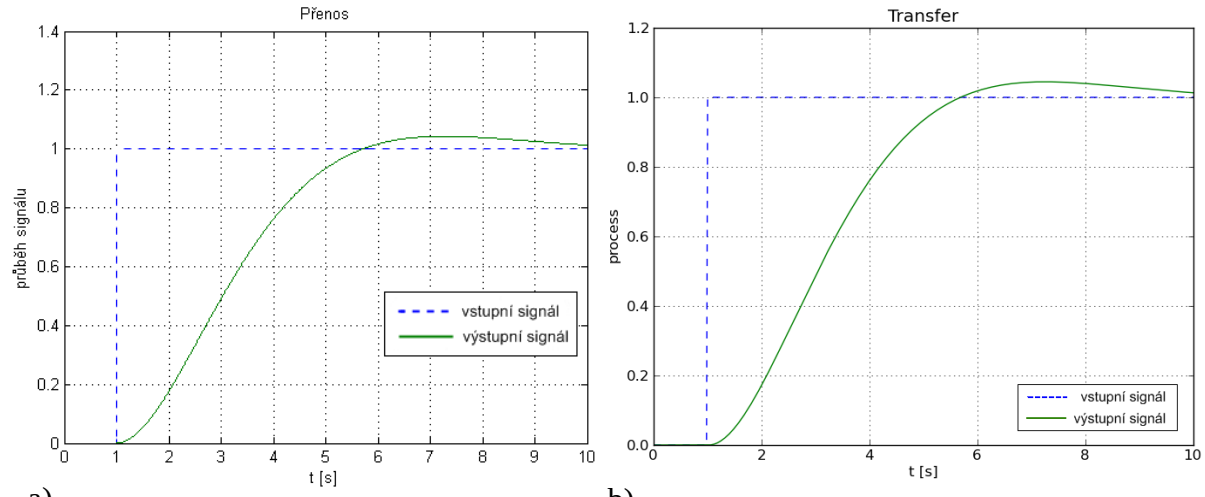
Obr. 6 – Průběh jednotkového skoku soustavou bez zpětné vazby. a) V Simulinku. b) V tomto programu.

4.4.4 Průběh jednotkového skoku soustavou se zpětnou vazbou

Pokud je potřeba simulovat přenos celého tohoto systému se zpětnou vazbou, postupujeme stejně jako v předcházejícím případě. Pouze u funkce *transfer* musíme zapnout zpětnou vazbu tím, že pátý parametr funkce změníme na hodnotu True. Na Obr. 7 je vidět porovnání výpočtu v Simulinku a tomto programu.

Kód v Pythonu:

```
Tf1 = [[2.515, -2.5], [1, -1],'z']     # přenos přes regulátor
Tf2 = [[1],[5, 8, 3]]                  # přenos přes systém

y = transfer(Tf1,Tf2,t,u,True)         # výpočet výstupu
plot(t,y)                              # vykreslení grafu
```



a) b)

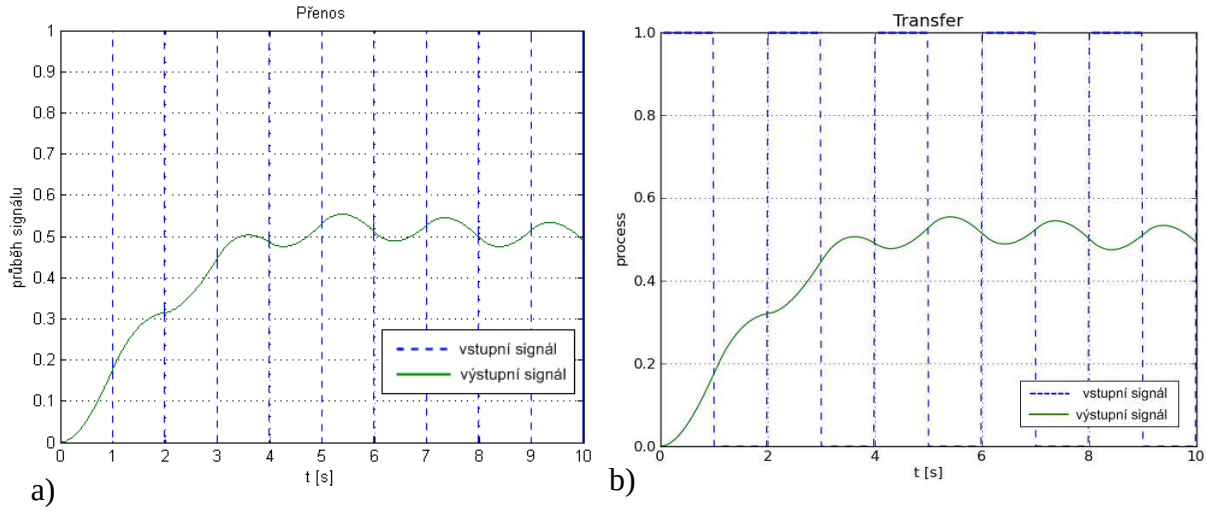
Obr. 7 – Průběh jednotkového skoku soustavou se zpětnou vazbou. a) V Simulinku. b) V tomto programu.

4.4.5 Průběh se vstupem ve tvaru obdélníka se zpětnou vazbou

Pokud je potřeba simulovat jiný typ vstupního signálu než jednotkový skok, pak lze použít například obdélníkový průběh s periodou 2s a délkou sepnutého stavu 1s. V tomto případě stačí místo příkazu *step* použít příkaz *pulse*. Na Obr. 8 je vidět porovnání výpočtu v Simulinku a tomto programu.

Kód v Pythonu:

```
u = pulse(t,2,1)  # vytvoření pole vstupů pulzním signálem
                  # s periodou 2s a délkou sepnutého stavu 1s
```



a) b)

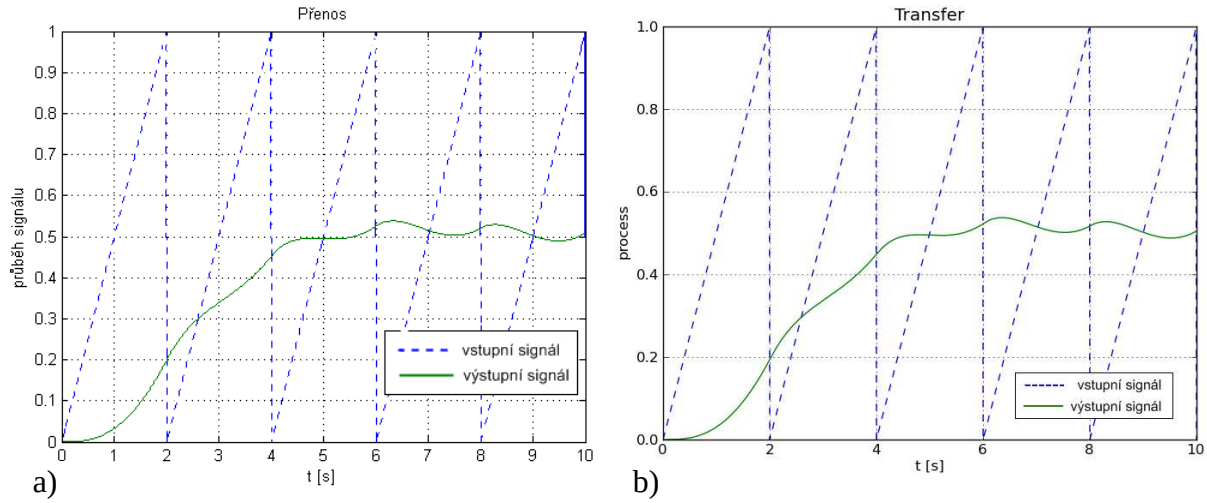
Obr. 8 – Průběh se vstupem ve tvaru obdélníka. a) V Simulinku. b) V tomto programu.

4.4.6 Průběh se vstupem ve tvaru pily se zpětnou vazbou

Další možnost je použít generování signálu ve tvaru pily tak, že u příkazu *pulse* zapneme ve čtvrtém parametru volbu *saw*. Na Obr. 9 je vidět porovnání výpočtu v Simulinku a v tomto programu.

Kód v Pythonu:

```
u = pulse(t,2,2,True) # vytvoření pole vstupů s pilovým signálem
                      # s periodou 2s a délkou sepnutého stavu 2s
```



a) b)

Obr. 9 – Průběh se vstupem ve tvaru pily. a) V Simulinku. b) V tomto programu.

4.4.7 Nelineární soustava - kyvadlo

V tomto programu lze také simulovat nelineární soustavu, například matematické kyvadlo, které popisuje rovnice

$$\ddot{\varphi} = -\frac{g}{l}\sin\varphi + u\,, \tag{8}$$

kde $\ddot{\varphi}$ je úhlové zrychlení, φ je úhel natočení, g je tíhové zrychlení, l je délka kyvadla a u je vstup.

Tato rovnice se dá snadno přepsat jako funkce do formátu pro Python. Budeme uvažovat následující počáteční podmínky, kdy je zadána počáteční výchylka φ kyvadla a jeho rychlost $\dot{\varphi}$:

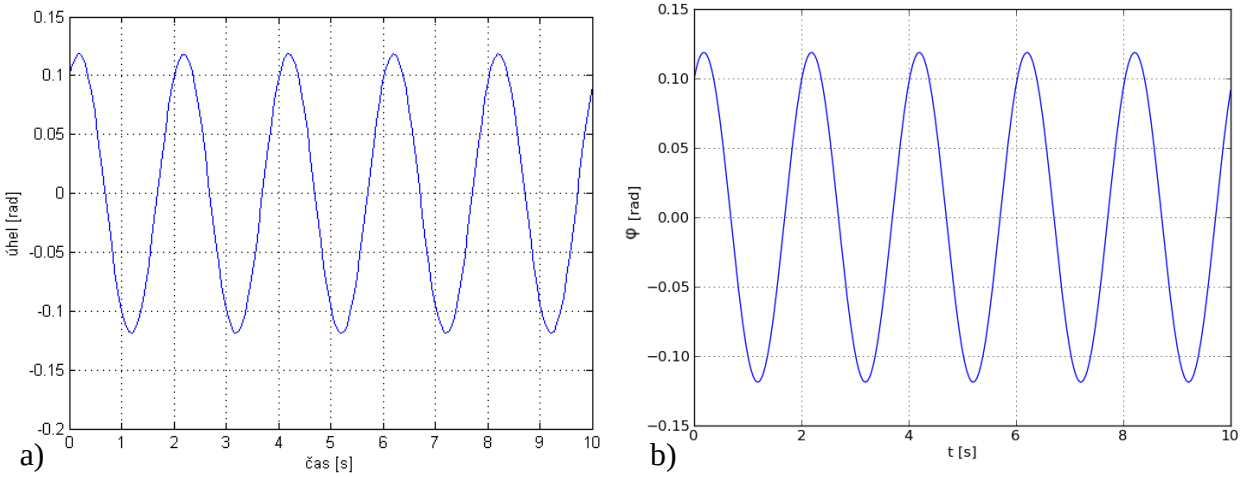
$$\varphi = 0{,}1 rad\,, \tag{9}$$

$$\dot{\varphi} = 0{,}2 rad \cdot s^{-1}. \tag{10}$$

Na Obr. 10 je vidět porovnání výpočtu v Simulinku a tomto programu.

Kód v Pythonu:

```
def kyvadlo(x,t,u):                    # definice funkce
    dx = [0.0] * 2                     # vytvoření výstupního vektoru

    g = 9.81                           # zadání konstant
    l = 1

    dx[0] = x[1]                       # zápis rovnic
    dx[1] = (-g/l)*sin(x[0]) + u[0]

x0 = [[0.0],[0.1,0.2]]                 # zadání počátečních podmínek
Tf1 = [[1],[1]]                        # jednotkový přenos přes regulátor
t = time(10,0.01)                      # vytvoření pole času
u = [0.0]*len(t)                       # vytvoření pole vstupů
y = transfer(Tf1,kyvadlo,t,u,False, x0) # výpočet průběhu
plot(t,y)                              # vykreslení grafu
```



a) b)

Obr. 10 – Kmitání kyvadla. a) V Simulinku. b) V tomto programu.

4.4.8 Kyvadlo – návrh řízení

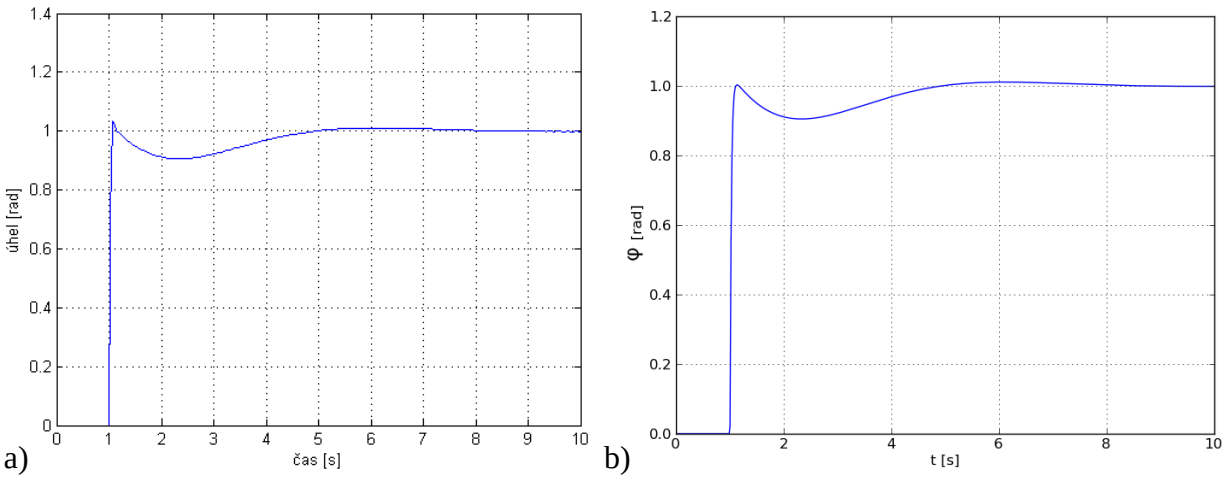
Pro kyvadlo z minulého příkladu lze navrhnout regulátor pomocí PID prvku obsaženém v programu. Na Obr. 11 je znázorněn jednotkový skok úhlu kyvadla s PID regulátorem, který má následující složky:

$$K_p = 40,\ K_i = 40, K_d = 40, \tag{11}$$

kde, K_p je konstanta proporcionálního členu, K_i konstanta integračního členu a K_d konstanta derivačního členu.

Kód v Pythonu:

```
x0 = [[0.0],[0.0,0.0]]              # zadání počátečních podmínek
PID = [40.0,40.0,40.0]              # zadání PID regulátoru

t = time(10,0.01)                   # vytvoření pole času
u = step(t,1)                       # vytvoření pole vstupů

y = transfer(PID,funkce,t,u,False, x0) # výpočet průběhu
plot(t,y)                           # vykreslení grafu
```



Obr. 11 – Simulace řízení kyvadla. a) V Simulinku. b) V tomto programu.

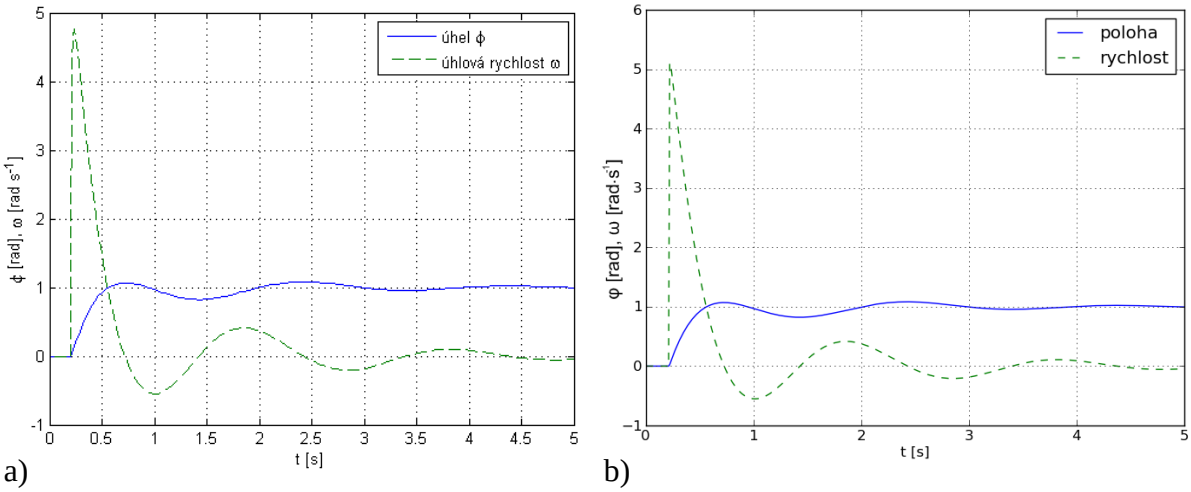
4.4.9 Kyvadlo – zobrazení úhlu natočení a rychlosti

V případě nutnosti je pro kyvadlo z příkladu 4.4.7 možné zobrazení dvoukanálového výstupu (nejen poloha, ale i rychlost kyvadla). Stačí vytvořit dvoukanálový signál pomocí funkce *step*, čímž celá soustava bude pracovat v dvoukanálovém režimu a na výstupu se pak objeví obě potřebné veličiny. Na Obr. 12 je znázorněna poloha a rychlost kyvadla při reakci na jednotkový skok nastávající v čase 0,2s s PID regulátorem, který má následující složky:

$$K_p = 10,\ K_i = 40, K_d = 5. \tag{12}$$

Kód v Pythonu:

```
x0 = [[0.0],[0.0,0.0]]          # zadání počátečních podmínek
PID = [10.0,40.0,5.0]           # zadání PID regulátoru

t = time(10,0.01)               # vytvoření pole času

u = step(t,[0.2,10])            # vytvoření dvoukanálového vstupu
y = transfer(PID,fx2,t,u,True,x0) # výpočet přenosu
plot(t,y)                       # vykreslení grafu
```



a) b)

Obr. 12 – Výstup 2 signálů z kyvadla. a) V Simulinku. b) V tomto programu.

4.4.10 Stejnosměrný motor

Dynamika stejnosměrného motoru je zachycena ve dvou diferenciálních rovnicích (13) a (14), které vycházejí z rovnic pro elektrickou rovnováhu kotvy a mechanickou rovnováhu momentů na hřídeli:

$$i' = \frac{1}{L}\left(U - Ri - k_m\omega\right), \tag{13}$$

$$\omega' = \frac{1}{J}\left(k_m i - M_0\right), \tag{14}$$

kde U je napájecí napětí, i je proud kotvy, R odpor vinutí kotvy, L vlastní indukčnost kotvy, M_0 zatěžovací moment, J moment setrvačnosti motoru, ω úhlová rychlost motoru a k_m je mechanická konstanta motoru.

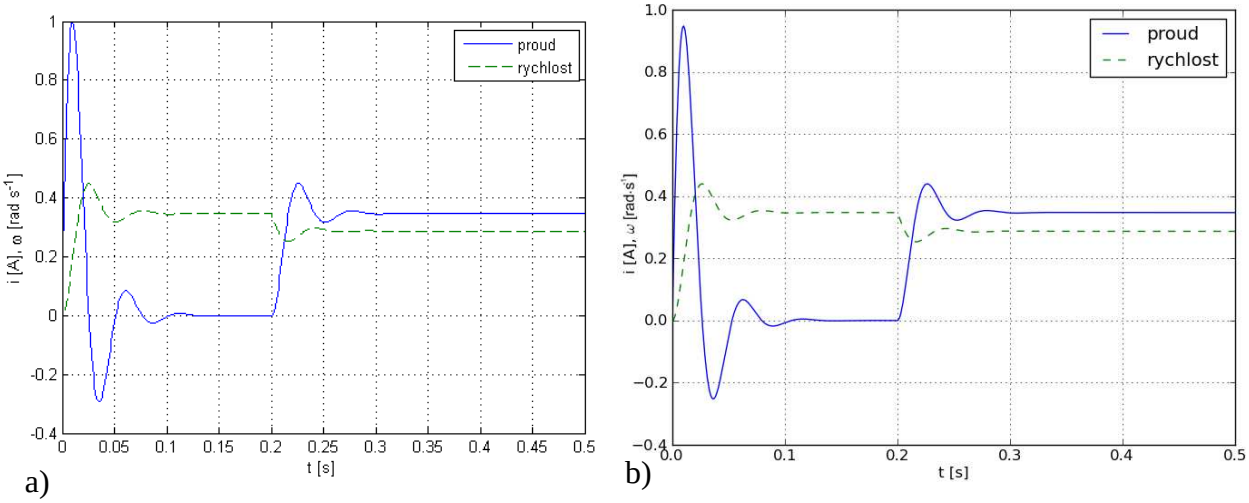
Aby byl na výstupu vidět průběh proudu i úhlové rychlosti, stačí pomocí příkazu *step* vytvořit pole dvou vstupů. První vstup můžeme zvolit jako napětí U, druhý jako zátěžný moment M_0. Následně je potřeba upravit rovnice (13) a (14) pro zápis funkce v jazyku Python. Simulace rozběhu motoru (viz Obr. 13) proběhla s následujícími parametry a počátečními podmínkami:

$$R = 0{,}5\Omega,\ L = 0{,}005H,\ k_m = 2{,}88,\ J = 0{,}1kgm^2,\ i_0 = 0A,\ \omega_0 = 0. \tag{15}$$

V čase $t = 0s$ bylo na vstup 1 přivedeno napětí $U = 1V$, a v čase $t = 0{,}2s$ byl na vstup 2 přiveden zátěžný moment $M_0 = 1Nm$.

Kód v Pythonu:

```
def Motor(x,t,u):                       # definice funkce
    dx = [0.0] * 2                      # vytvoření výstupního vektoru

    km = 2.88                           # zadání konstant
    R = 0.5
    J = 0.1
    L = 0.005

    dx[0] = (u[0]-R*x[0]-km*x[1])/L # zápis rovnic
    dx[1] = (km*x[0]-u[1])/J

x0 = [[0.0],[0.0]]                      # zadání počátečních podmínek
PID = [1,0,0]                           # jednotkový přenos přes regulátor

t = time(0.5,0.001)                     # vytvoření pole časů s krokem
u = step(t,[0.0,0.2])                   # vytvoření pole 2 vstupů

y = transfer(PID,Motor,t,u,False,x0) # výpočet průběhu
plot(t,y)                               # vykreslení grafu
```



a) b)

Obr. 13 – Rozběh stejnosměrného motoru. a) V Simulinku. b) V tomto programu.

4.4.11 Řízení polohy stejnosměrného motoru

Aby bylo možné řídit polohu motoru, je nutné k rovnicím dynamiky motoru (13) a (14) přidat rovnici natočení motoru

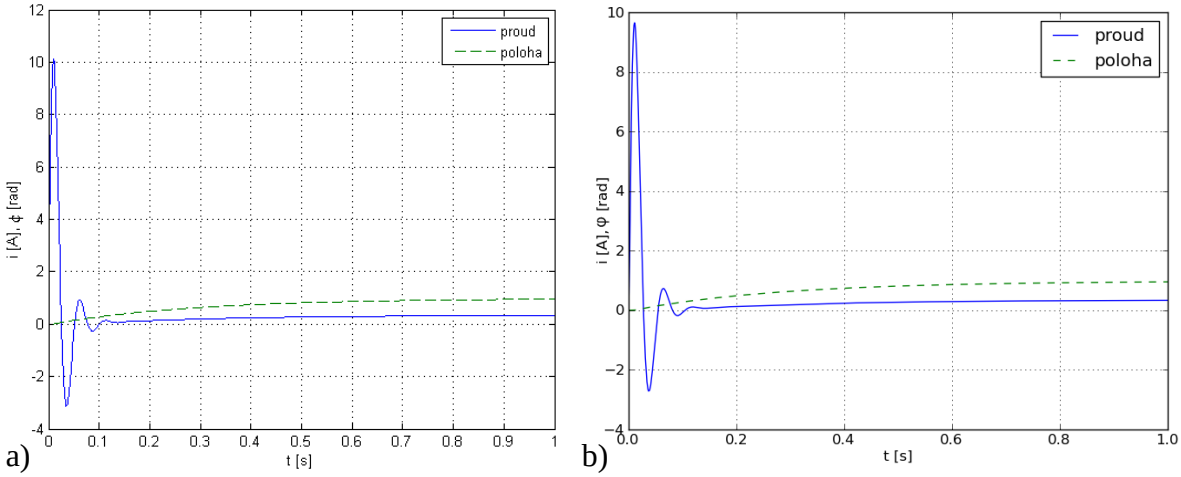
$$\dot{\varphi} = \omega, \tag{16}$$

a všechny tyto rovnice přepsat do tvaru pro jazyk Python. Aby byla na výstupu vidět nejen poloha, ale i proud, byl vytvořen dvouvstupový systém, u něhož je používán pouze druhý vstup znázorňující polohu motoru. Ten je pak řízen pomocí zpětné vazby a PID regulátoru. Parametry motoru zůstaly stejné, jako v předešlém příkladu 4.4.10, jen zatěžný moment M_0 byl zvolen jako konstantní s hodnotou $M_0 = 1Nm$.

Kód v Pythonu:

```
def Motor(x,t,u):                       # definice funkce
    dx = [0.0] * 3                      # vytvoření výstupního vektoru

    km = 2.88                           # zadání konstant
    R = 0.5
```

```
    J = 0.1
    L = 0.005
    M0 = 1

    dx[0] = (u[1]-R*x[0]-km*x[2])/L# rovnice - proud
    dx[1] = x[2]                   #         - poloha
    dx[2] = (km*x[0]- M0)/J        #         - rychlost

x0 = [[0.0],[0.0]]                 # zadání počátečních podmínek
PID = [10,0,0]                     # jednotkový přenos přes regulátor

t = time(1,0.001)                  # vytvoření pole časů s krokem
u = step(t,[1.0,0.0])              # vytvoření pole 2 vstupů

y = transfer(PID,Motor,t,u,True,x0)# výpočet průběhu
plot(t,y)                          # vykreslení grafu
```

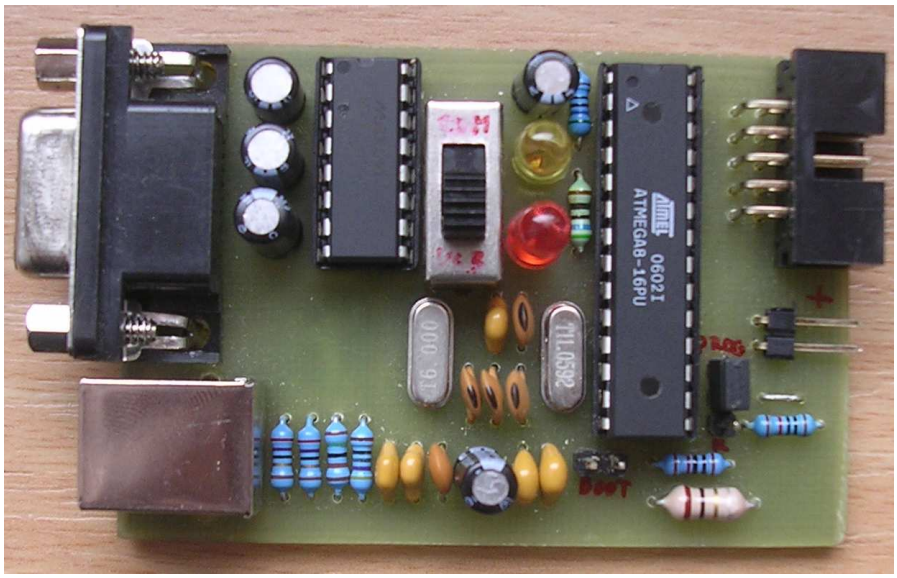


Obr. 14 – Řízení polohy stejnosměrného motoru. a) V Simulinku. b) V tomto programu

4.5 Řízení modelu pomocí mikrokontroléru

Následující příklady ukazují, jak lze vytvořit z mikrokontroléru regulátor řídící model po sériové lince. Toto řešení nelze použít v praxi. Sériová linka je pro řízení velice pomalá a nelze ji použít pro regulaci v reálném čase. K demonstraci funkcí programu z této práce však plně postačuje.

Původním záměrem této práce bylo naprogramovat rychlejší šestnáctibitový mikrokontrolér dsPIC. V důsledku jeho složitější dokumentace a dalších komplikací byl nakonec pro řízení modelu v nereálném čase použit osmibitový mikrokontrolér ATmega8 (viz. Obr. 15) od firmy Atmel.



Obr. 15 – Mikrokontrolér ATmega8. Deska původně sloužila jako programátor pro mikrokontroléry AVR, po přeprogramování slouží jako regulátor.

4.5.1 Generování C-kódu s datovým typem double

Pokud je potřeba uřídit polohu kyvadla stejného jako v příkladu 4.4.7 a je-li na to zvolen regulátor stejný jako v příkladu 4.4.9, pak třeba nejprve pro tento regulátor vygenerovat podprogram C-kódu pomocí funkce *CompileC*.

Kód v Pythonu:

```
PID = [10.0,40.0,5.0]          # zadání PID regulátoru
T = 0.01                       # zadání času 1 kroku v sekundách
CreateC('C:\pid', PID, T, typ = 'double')
      # vygenerování podprogramu PID reg. do souboru 'C:\pid.c',
      # který pracuje s datovým typem double s krokem 0.01s
```

Vygenerovaný kód pak vypadá následovně, zde je pro vysvětlení ještě doplněn komentáři:

```
/*********************************************************************
* Regulator PID                                                      *
*********************************************************************/

/************* DEFINE CONSTANTS ***************/

#define Kp 10.0              // konstanty pro výpočet
#define Ki 0.4
#define Kd 500
#define T 0.01

/************* GLOBAL VARIABLES ***************/

double P, I, D, S;           // proměnné

/************* PROCEDURES *********************/

void init_reg(void)          // inicializace proměnných
{
      P = 0;
      I = 0;
      D = 0;
      S = 0;
}
```

```
double calculate(double value)      // výpočet regulátoru
{
      P = value * Kp;               // P složka
      I = I + Ki * value;           // I složka
      D = Kd * (value - S);         // D složka
      S = value;                    // zápis vstupu pro D složku
      return(P + I + D);            // součet
}
```

4.5.2 Naprogramování mikrokontroléru

Vygenerovaný podprogram je nutné přidat do hlavního programu pro mikroprocesor. Ten musí mít takovou strukturu, aby načetl vstupní hodnotu, přepočítal ji pomocí tohoto podprogramu a vyslal na výstup. V tomto případě je hodnota načítána ve formě řetězce ze sériového portu, pak je převedena z řetězce na číslo, přepočítána, znovu převedena na řetězec a vyslána po sériovém portu zpět. Hlavní program pak vypadá následovně:

```
void main(void)
{
      char s[16]; int i; double f; // deklarace proměnných

      uart_init();                 // inicializace sériového rozhraní
      init_reg();                  // inicializace regulátoru

      while(1)                     // hlavní program
      {
            if((UCSRA & (1 << RXC)) == 1 ) // pokud přijat znak
            {
                  i = 0;
                  s[i++] = UDR;              // přijmout do proměnné
                  while (s[i-1] != '\n')
                  {                 // dokud nebude ukončovací znak
                        while ((UCSRA & (1 << RXC)) == 0) {};
                        s[i++] = UDR;        // přijmout do proměnné
                  }
                  s[i-1] = '\0';             // ukončit řetězec
                  f = atof(s);               // převést na číslo
                  f = calculate(f);          // výpočet
                  sprintf(s, "%.8f", f);     // převést na řetězec
                  putStr(s);                 // odeslat řetězec
            }
      }
}
```

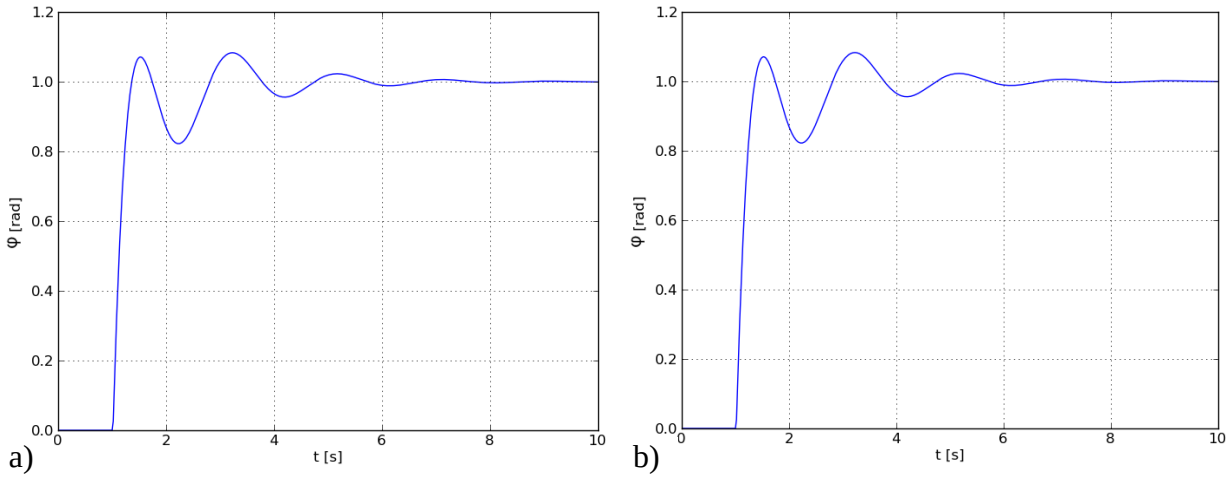
Celý kód lze zkompilovat v prostředí AVR Studio[23] pomocí kompilátoru AVR-GCC a zkompilovaný soubor nahrát do mikrokontroléru.

4.5.3 Regulace modelu pomocí mikrokontroléru

Naprogramovaným mikrokontrolérem můžeme řídit model kyvadla z příkladu 4.4.7. Stačí u funkce *transfer* dosadit do prvního bločku regulátoru název sériového portu, na který je mikrokontrolér připojen, v tomto případě COM3, do druhého bločku modelu zadat funkci kyvadla a výstup funkce následně vykreslit. Výsledný průběh je možné porovnat s výpočtem pomocí simulovaného PID regulátoru. Na Obr. 16 je vidět, že se tyto průběhy neliší.

Kód v Pythonu:

```
x0 = [[0.0],[0.0,0.0]]              # počáteční podmínky
t = time(10, 0.01)                  # vytvoření pole času
u = step(t,1)                       # vytvoření jednot. Skoku
y = transfer(['COM3'],kyvadlo,t,u,True,x0) # výpočet průběhu
plot(t,y)                           # vykreslení
```



a) b)

Obr. 16 – Regulace modelu kyvadla. a) Mikrokontrolér. b) Simulovaný PID regulátor

4.5.4 Regulace modelu pomocí mikrokontroléru s datovým typem integer

Pro zrychlení délky výpočtu regulátoru je lepší používat celočíselný datový typ integer. Při programování mikrokontroléru se postupuje prakticky stejně jako v případě práce s datovým typem double, pouze u funkce *CompileC* se změní parametr *typ* na *‚int‘*. Dále je třeba dbát na to, aby parciální složka, součin integrační složky s délkou kroku a podíl derivační složky s délkou kroku byla celá čísla. Tomuto požadavku odpovídá např. regulátor obsahující tyto složky:

$$K_p = 40,\ K_i = 400, K_d = 40. \tag{17}$$

Příklad:

```
PID = [40, 400,40]              # zadání PID regulátoru
T = 0.01                        # zadání času 1 kroku v sekundách
CreateC(‘C:\pid‘, PID, T, typ = 'int')
```

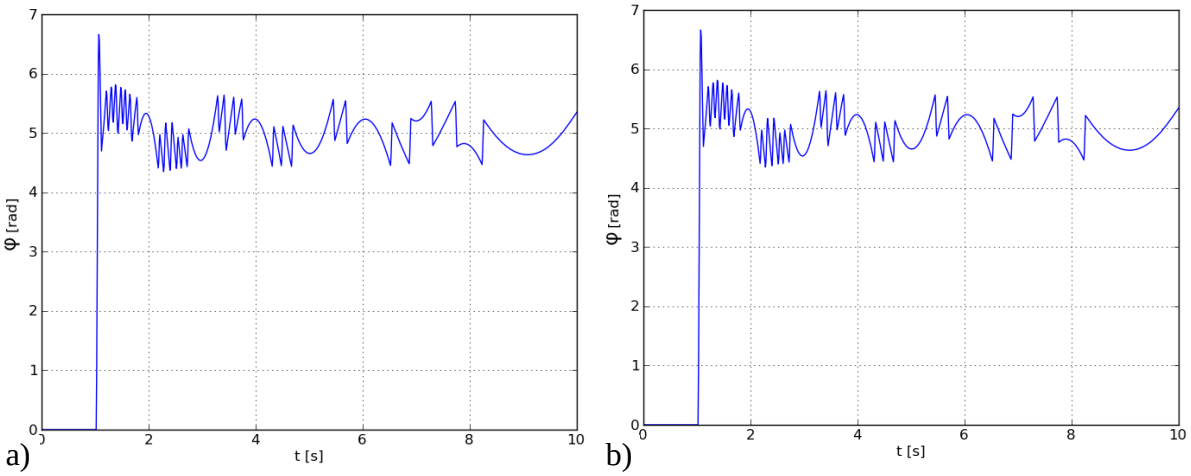
Po naprogramování mikrokontroléru je možné zkusit řídit model kyvadla tímto celočíselným regulátorem. Je však nutné zapnout ve funkci *transfer* u bločku s regulátorem práci s celými čísly tak, že za parametr *typ* dosadíme hodnotu *‘int‘*. Na Obr. 17 je vidět porovnání výsledků řízení celočíselným regulátorem a simulací tohoto regulátoru.

Kód pro simulaci PID regulátoru v Pythonu:

```
PID = [40, 400,40]              # zadání PID regulátoru
y = transfer(PID,kyvadlo,t,u,True,x0,False,[],['int','float'])
plot(t,y)                       # vykreslení
```

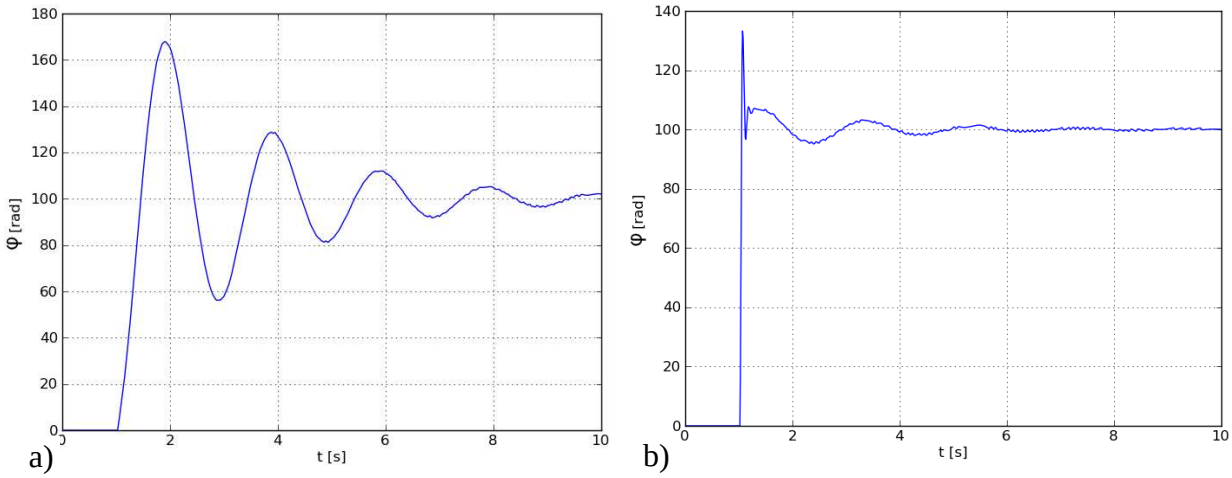
Kód pro simulaci mikrokontroléru v Pythonu:

```
y = transfer([‘COM3‘],kyvadlo,t,u,True,x0,False,[],['int','float'])
plot(t,y)                       # vykreslení
```



Obr. 17 – Řízení celočíselným regulátorem. a) Mikrokontrolér. b) Simulátor.

Oba průběhy řízení pro skok při natočení 5 radiánů se neliší. Zlom nastává při větších natočeních, například při 100 radiánech, kdy si simulátor se skokem poradil o mnoho lépe než mikrokontrolér, kterému s největší pravděpodobností přetekl integrační zásobník. Porovnání průběhů je na Obr. 18.



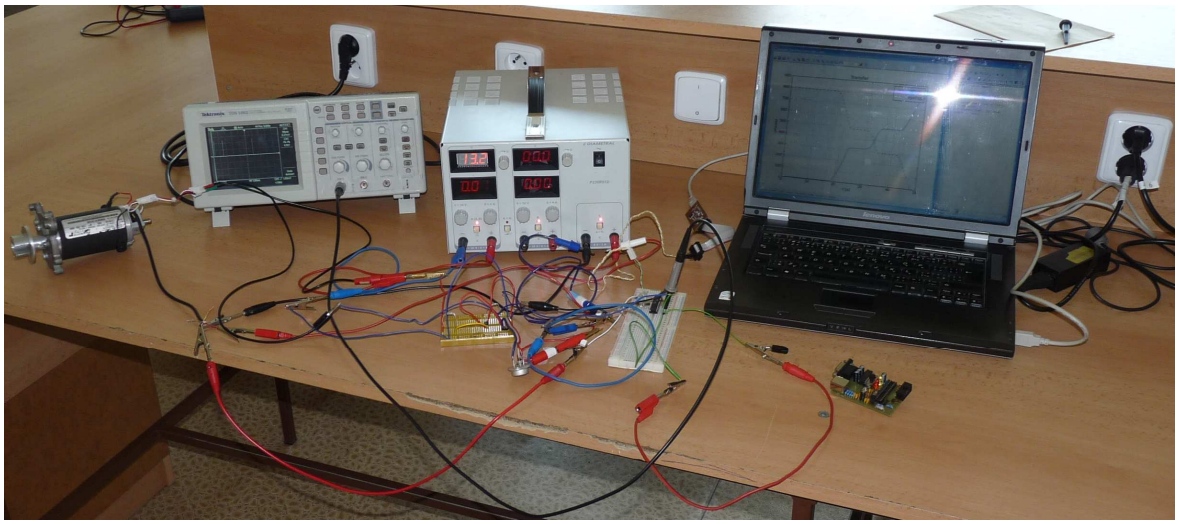
Obr. 18 – Řízení celočíselného regulátoru. a) Mikrokontrolér. b) Simulátor.

4.6 Řízení reálné aplikace pomocí mikrokontroléru

Vyzkoušení řízení reálné aplikace pomocí mikrokontroléru fungujícího jako PID regulátor probíhalo v laboratoři měření Střední průmyslové školy Třebíč (viz Obr.19). Na řízení byl použit stejnosměrný motor s enkodérem, který vytvářel 512 impulzů na jedno otočení rotoru motoru. Tyto impulzy byly přivedeny do mikrokontroléru a sloužily jako zpětná vazba pro řízení polohy motoru. Mikrokontrolér řídil motor pomocí PWM signálu, který byl převáděn výkonovým členem na velikost napětí.

Kvůli složitosti dekódování směru otáčení z enkodéru a chybějícímu výkonovému členu podporujícímu změnu polarity napětí byla poloha motoru pro zjednodušení regulace řízena pouze jedním směrem otáčení.

V následujících kapitolách jsou rozebrány příklady regulace stejnosměrného motoru mikrokontrolérem.



Obr. 19 – Laboratorní pracoviště

4.6.1 Regulace natočení motoru pomocí zpětné vazby

V tomto případě mikrokontrolér načítal impulsy z enkodéru a v časovém intervalu 0,01s porovnával s požadovanou hodnotou natočení 800 impulsů. Podle toho měnil mikrokontrolér střídu PWM signálu a tím řídil napětí přivedené na motor.

Aby bylo možné kontrolovat průběh řízení, byla pro zjištění hodnot vstupů a výstupů mikrokontroléru použita funkce *status*. Její popis je v kapitole 4.3.

Struktura C kódu pro mikrokontrolér vypadala následovně:

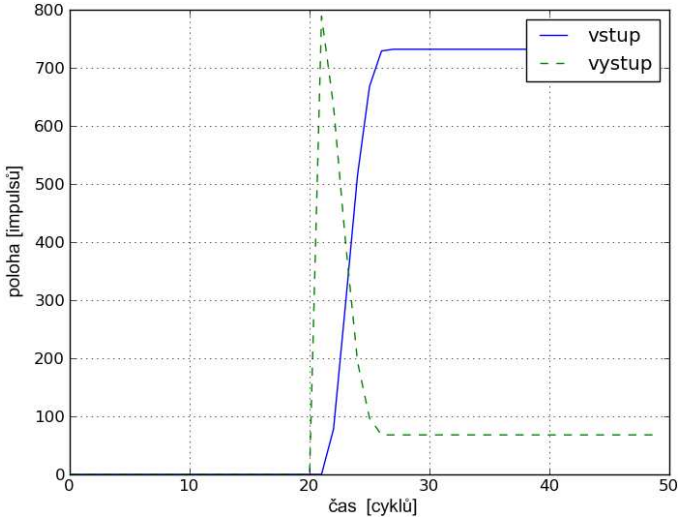
```
int vystup, poloha;                 // deklarace globálních proměnných

void main(void)                     // hlavní program
{
      char c, s[16]; int i;         // deklarace lokálních proměnných
      uart_init();                  // inicializace sériového portu
      init_int();                   // inicializace počítání impulsů
      init_PWM();                   // inicializace generování PWM
      init_Time();                  // inicializace odpočítávání 0.01s
      sei();                        // povolení přerušení

      while(1)
      {
            c =getChar();           // přijmout znak
            if (c == 'u')           // pokud vstup
            {
                  itoa(poloha, s, 10); // převést číslo na řetězec
                  sendString(s);       // poslat hodnotu vstupu
            }
            else if (c == 'y')          // pokud výstup
            {
                  itoa(vystup, s, 10); // převést číslo na řetězec
                  sendString(s);       // poslat hodnotu výstupu
            }
      }
}

ISR(TIMER2_COMP_vect)               // přerušení hl. prog. každých 0.01s
{
      char s[16];                   // deklarace lokálních proměnných
      vystup = 800 - poloha;        // výp. rozd. požad. a skut. hodnoty
      if (vystup < 0) {vystup = 0;} // pokud záporná, oříznout na 0
      if (vystup > 1023) {vystup = 1023;} // kladná, oříz. na maximum
      OCR1A = vystup;               // poslat hodnotu na generování PWM
}
```

```
ISR(INT0_vect)                    // přerušení při vzestupné hraně
{                                 // signálu z enkoderu
      poloha += 1;                // přičtení impulzu k poloze
}
```

V Pythonu pak stačilo pomocí funkce *status* načíst 50 hodnot z mikrokontroléru připojeného na port COM3 a vykreslit průběh polohy rotoru pomocí funkce *plot*. Jak je vidět na Obr. 20, poloha nedosáhla požadované hodnoty 800 impulzů.

Kód v Pythonu:

```
y = status('COM3', 50)            // zjišťování 50 hodnot z mikrokont.
t = range(50)                     // vytvoření hodnot pro osu x
plot(t,y)                         // vykreslení grafu
```



Obr. 20 – Motor pouze se zpětnou vazbou

4.6.2 Regulace motoru pomocí PID regulátoru pracujícím s čísly typu integer

V případě 4.6.1 poloha motoru nedosáhla požadované hodnoty. Proto by bylo vhodné použít PID regulátor. V tomto příkladu je použit pouze s proporcionální složkou o velikosti K_p = 5. Podprogram regulátoru je vygenerován pomocí funkce *CreateC* a přidán do hlavního programu tak, aby se hodnota polohy před odesláním do PWM přepočítala.

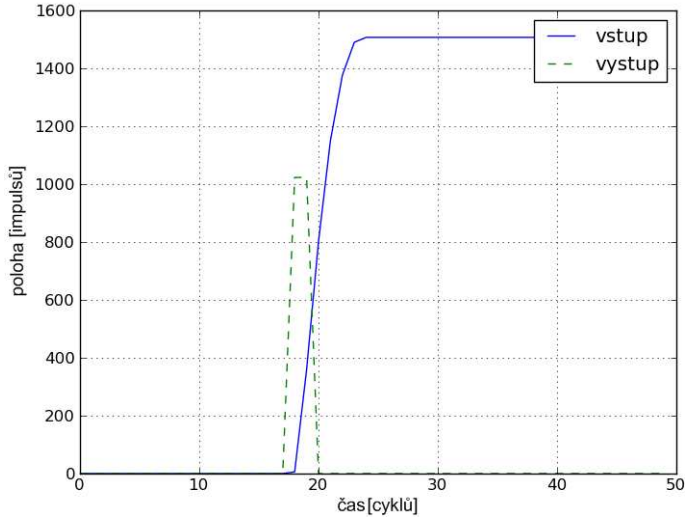
Jak je vidět na Obr. 21, má motor s danou proporcionální složkou regulátoru od požadované hodnoty 800 impulzů velký překmit.

Kód v Pythonu:

```
PID = [5,0,0]
T = 0.01
CreateC('xxx', PID, T,'int')
```

Změněné přerušení v C kódu:

```
ISR(TIMER2_COMP_vect)             // přerušení hl. prog. každých 0.01s
{
   int x; char s[16];   // deklarace lokálních proměnných
   x = 800 - poloha;              // výp. rozd. požad. a skut. Hodnoty
   vystup = calculate(x);         // přepočet PID regulátoru
   if (vystup < 0) {vystup = 0;} // pokud záporná, oříznout na 0
   if (vystup > 1023) {vystup = 1023;} // kladná, oříz. na maximum
   OCR1A = vystup;                // poslat hodnotu na generování PWM
}
```

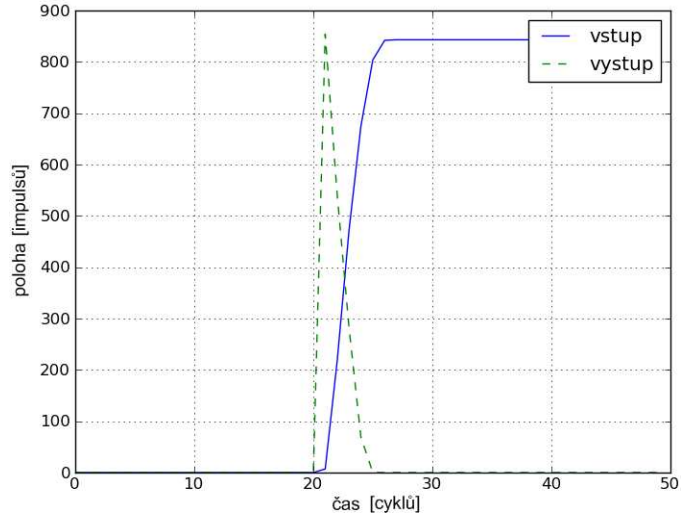


Obr. 21 – Motor se zpětnou vazbou a PID regulátorem pracujícím s čísly typu integer

4.6.3 Regulace motoru pomocí PID regulátoru pracujícím s čísly typu double

V případě 4.6.2 poloha motoru přesáhla požadovanou hodnotu. Bylo by vhodné použít PID regulátor pracující s menší hodnotou proporcionální složky. Postup práce je stejný jako v minulém případě. Jediným rozdílem je použití regulátoru s proporcionální složkou hodnoty K_p = 1,2. Tato hodnota je ovšem desetinná, proto je použit pro výpočet datový typ double. Přepočítaná hodnota se pak musí převést zpět na hodnotu integer.

Jak je vidět na Obr. 22, má motor s touto proporcionální složkou regulátoru oproti případu 4.6.2 relativně malý překmit.

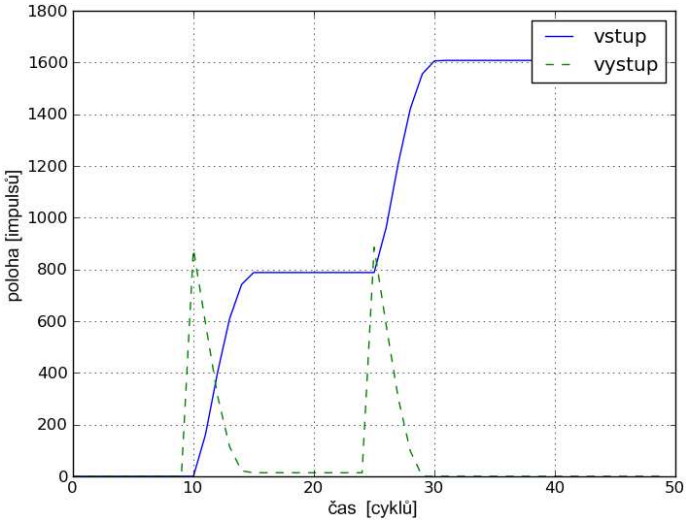


Obr. 22 – Motor se zpětnou vazbou a PID regulátorem pracujícím s čísly typu double

Ale aby se i tento překmit odstranil, byl použit PID regulátor se složkami:

$$K_p = 2\,,\ K_i = 0\,,\ K_d = 0{,}02\,. \tag{18}$$

Průběh, kdy od desátého cyklu je požadovaná hodnota polohy motoru 800 impulzů a od dvacátého pátého cyklu 1600 impulzů, je znázorněn na Obr. 23. Řízení polohy už je velmi přesné.



Obr. 23 – Vyladěný PID regulátor

5 ZÁVĚR

V této práci byl vytvořen simulátor pro jednoduchý model řízené soustavy, který obsahoval regulátor, zpětnou vazbu, šum a generátor referenční trajektorie. Dynamiku tohoto modelu lze definovat pomocí explicitní ODE. Funkce simulátoru byla znázorněna na příkladu matematického kyvadla a na řízení polohy stejnosměrného motoru a porovnána s výpočtem pomocí prostředí Matlab/Simulink. Všechny sledované nasimulované průběhy se shodovaly, což by mělo svědčit o správnosti naprogramování simulátoru. Rozdíl byl pozorován u některých velmi strmých průběhů. V prostředí Simulink byla velikost signálu nepatrně vyšší. To mohlo být způsobeno rozdílným výpočtem integrace, například jinou volbou kroku.

Dynamika modelu i regulátoru může být nahrazena komunikací po sériové lince s libovolným zařízením. Tím je možné odsimulovat správné naprogramování výpočtu diskrétního regulátoru přímo na mikrokontroléru.

Práci je možné také využít pro zjednodušení programování regulátoru do mikrokontrolérů. Toho lze dosáhnout pomocí převodu diskrétní přenosové funkce nebo PID regulátoru jako podprogram pro C-kód, který může pracovat buď s čísly s plovoucí desetinnou čárkou nebo s čísly celými.

Tento podprogram byl při aplikaci přidán do programu vytvořeného pro mikrokontrolér Atmel ATmega8. Následně byla odsimulována jeho správná funkčnost na modelu vytvořeném v simulátoru i na reálném motoru.

6 LITERATURA A ODKAZY

[1] HARMS, Daryl; MCDONALD, Kenneth. *Začínáme programovat v jazyce Python*. 1. vydání. Brno: Computer Press, 2003. 456 s. ISBN 80-7226-799-X

[2] SKLICKÝ, Jiří. *Teorie řízení*. 2. vydání, Brno: VUT FEKT, 2002. 98 s.

[3] PYTHON SOFTWARE FOUNDATION. *Python*. [online]. 2010-05-10.
Dostupné z < http://www.python.org/ >

[4] PYTHON SOFTWARE FOUNDATION. *Python install files*. [online]. 2010-05-10.
Dostupné z < http://www.python.org/ftp/python/2.6.5/python-2.6.5.msi >

[5] NUMPY DEVELOPERS. *NumPy*. [online]. 2010-05-10.
Dostupné z < http://numpy.scipy.org/ >

[6] NUMPY DEVELOPERS. *NumPy install files*. [online]. 2010-05-10.
Dostupné z < http http://sourceforge.net/projects/numpy/files/ >

[7] NUMPY COMMUNITY. *NumPy User Guide* [online]. 2010-04-15.
Dostupné z < http://docs.scipy.org/doc/numpy/numpy-user.pdf >

[8] NUMPY COMMUNITY. *NumPy Reference Guide* [online]. 2010-04-15.
Dostupné z < http://docs.scipy.org/doc/numpy/numpy-ref.pdf >

[9] SCIPY DEVELOPERS. *SciPy*. [online]. 2010-05-10.
Dostupné z < http://scipy.org/ >

[10] SCIPY DEVELOPERS. *SciPy install files*. [online]. 2010-05-10.
Dostupné z < http://sourceforge.net/projects/scipy/files/ >

[11] SCIPY COMMUNITY. *SciPy Reference Guide* [online]. 2010-04-15.
Dostupné z < http://docs.scipy.org/doc/scipy/scipy-ref.pdf >

[13] DALE, Darren a kolektiv. *Matplotlib*. [online]. 2010-05-10.
Dostupné z < http://matplotlib.sourceforge.net/ >

[14] DALE, Darren a kolektiv. *Matplotlib documentation*. [online]. 2010-05-10.
Dostupné z < http://matplotlib.sourceforge.net/Matplotlib.pdf >

[15] DALE, Darren a kolektiv. *Matplotlib install files*. [online]. 2010-05-10.
Dostupné z < http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-0.99.1/ >

[16] LIECHTI, Chris. *pySerial*. [online]. 2010-05-10.
Dostupné z < http://pyserial.sourceforge.net/ >

[17] LIECHTI, Chris. *pySerial install files*. [online]. 2010-05-10.
Dostupné z < http://sourceforge.net/projects/pyserial/files/ >

[18] WIKIPEDIA. *Mikrokontrolér PIC*. [online]. 2010-05-10.
Dostupné z < http://cs.wikipedia.org/wiki/Mikrokontrolér_PIC >

[19] MICROCHIP. *dsPIC33FJ128MCX02/X04 Data Sheet*. [online]. 2010-05-10.
Dostupné z < http://ww1.microchip.com/downloads/en/DeviceDoc/70291D.pdf >

[20] MATHWORKS. *MATLAB/Simulink*. [online]. 2010-05-10.
Dostupné z < http://www.mathworks.com/ >

[21] EATON, J.W. a kolektiv. *Octave*. [online]. 2010-05-10.
Dostupné z < http://www.gnu.org/software/octave/ >

[22] TIŠNOVSKÝ, Pavel. [online]. 2006-05-24
Dostupné z < http://www.root.cz/clanky/fixed-point-arithmetic/ >

[23] ATMEL. AVR Studio. [online] 2010-05-21.
Dostupné z < http://www.atmel.com/dyn/products/tools_card.asp?tool_id=2725 >

[24] HEROUT, Pavel. *Učebnice jazyka C*. 4.vydání. České Budějovice: Kopp, 2007. 271 s. ISBN 80-7232-220-6

[23] MM PRŮMYSLOVÉ SPEKTRUM. Rotační snímače (enkodéry). [online]. 2010-05-26.
Dostupné z < http://www.mmspektrum.com/clanek/rotacni-snimace-enkodery >

7 SEZNAM POUŽITÝCH SYMBOLŮ

φ – úhlové zrychlení $[\mathrm{rad}]$

ω – úhlová rychlost $[\mathrm{rad} \cdot \mathrm{s}^{-1}]$

g – tíhové zrychlení $[\mathrm{m} \cdot \mathrm{s}^{-2}]$

l – délka $[\mathrm{m}]$

i – proud $[\mathrm{A}]$

U – napětí$[\mathrm{V}]$

R – elektrický odpor$[\Omega]$

L – indukčnost$[\mathrm{H}]$

k_m – konstanta motoru, která je dána konstrukčním provedením

M – moment$[\mathrm{Nm}]$

M_0 – zatěžovací moment$[\mathrm{Nm}]$

J – moment setrvačnosti$[\mathrm{kg} \cdot \mathrm{m}^2]$

t – čas $[\mathrm{s}]$

K_p – proporcionální složka PID regulátoru

K_i – integrační složka PID regulátoru

K_d – derivační složka PID regulátoru

kbps – sériová datová rychlost – kilobit za sekundu

explicitní ODE – explicitně vyjádřená soustava diferenciálních rovnic

PID – proporcionálně integračně derivační regulátor

PWM – pulzně šířková modulace

ASCII – kódy pro znaky v digitální technice

Enkodér – převodník převádějící rotační pohyb na číslicové impulzy[25]

Datové typy:

- Float – datový typ s plovoucí desetinnou čárkou
- Double – datový typ s plovoucí desetinnou čárkou se zvýšenou přesností
- Fixed point – datový typ s pevnou desetinnou čárkou
- Integer – celočíselný datový typ
- Boolean – datový typ s dvěmi možnostmi – pravda/nepravda (True/False)
- Ndarray – vícerozměrné pole z knihovny NumPy